Jarno Lahti

Mobiilipelin tekninen toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

9.5.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jarno Lahti Mobiilipelin tekninen toteutus 35 sivua + 1 liite 31.3.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Filosofian maisteri Teemu Saukonoja

Insinöörityön tavoitteena oli toteuttaa Suomalaiselle pelifirmalle mobiilipelin SumTower tekninen toteutus. Projekti toteutettiin Libgdx-sovelluskehyksellä ja ohjelmointikielenä projektissa käytettiin javaa. Insinöörityön tarkoituksena oli kehittää omaa osaamistani mobiilipelien kehityksessä ja samalla saada kokemusta työskentelystä mobiilipelien parissa.

Työn teoriaosuus on koottu verkosta löytyvistä tietolähteistä ja alan kirjallisuudesta. Teoriaosuudessa tutustutaan mobiilipelien historiaan, eri työkaluihin ja ohjelmointikieliin, joita mobiilipelien tekemiseen käytetään ja tarkastellaan tarkemmin Libgdx-sovelluskehyksen ominaisuuksia.

Käytännön osuus koostuu projektin pelilogiikan ratkaisujen tarkastelusta ja niiden selkokielelle avaamisesta lukijalle.

Projektille asetetut tavoitteet saatiin täytettyä, ja tuote saatiin julkaistua android-laitteille. Pelin kehitystä päätettiin jatkaa. Peli on ladattavissa Google play-sovelluskaupasta.

Avainsanat	Java, mobiilipeli, Libgdx

Author(s) Title Number of Pages Date	Jarno Lahti Technical implementation of a mobile game 35 pages + 1 appendixces 31 March 2016
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor(s)	Teemu Saukonoja, M.Sc.

The goal of this study was to implement technical the execution of a mobile game called SumTower for a Finnish game company. The project was implemented by utilizing the Libgdx framework and the Java programming language. The purpose was to learn about mobile game development and at the same time to gain experience of working with mobile games.

The theoretical part of this thesis introduces the history of mobile games, various tools and programming languages that are used to make a mobile game. The features of the Libgdx framework are also analyzed. In addition, the thesis reviews the project's game logic solutions.

The objectives set for the project were completed, and the product was released for Android devices. The game can be downloaded from the Google Play app store. The development of the game will also continue.

Keywords	Java, mobile games, Libgdx

Sisällys

Lyhenteet

FPS	First person shooter. Peligenre, jossa pelaaja kokee pelin pelattavan hahmon näkökulmasta.
APK	Android application package. Tiedostomuoto jota Android-käyttöjärjestelmä käyttää applikaatioiden jakamiseen ja asentamiseen.
API	Application Program Interface. Joukko työkaluja ja sääntöjä joita hyödynnetään sovelluskehityksessä.
HTTP	Hypertext Transfer Protocol. Protokolla tiedon lähetykseen ja vastaanottamiseen.
HTTPS	Hypertext Transfer Protocol Secure. Suojattu protkolla tiedon lähetykseen ja vastaanottoon.

1 Johdanto

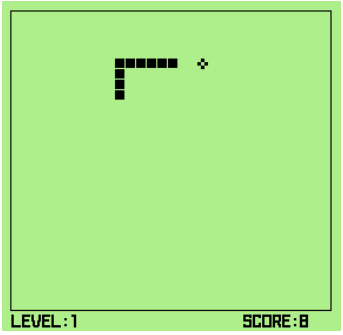
Opinnäytetyön aiheena on Taphold Oy:lle tuotetun mobiilipelin tekninen toteutus. Taphold Oy on kuuden opiskelijan perustama peliyhtiö, joka keskittyy mobiilipelien tuottamiseen ja julkaisuun. Taphold Oy:n ensimmäinen mobiilipeli SumTower on julkaistu Androidille Googlen Play-kaupassa, joka on Android-sovelluksille tarkoitettu kauppapaikka. SumTower on pulmapeli, jossa yhdistyy matematiikka- ja match-3-henkinen pelimekaniikka.

Tässä raportissa käyn läpi SumTowerin pelilogiikan teknistä toteutusta ja kerron tekniikoista sekä työkaluista, joita pelin tuottamiseen käytettiin. Selitän myös lyhyesti mobiilipelien historiasta sekä käyn läpi java-ohjelmointikieltä ja olio-ohjelmointia. Tavoitteena opinnäytetyössä on tarkastella mobiilipelikehitystä libgdx-sovelluskehyksellä ja kehittää omaa osaamista mobiilipelikehityksessä.

2 Mobiilipelit

Mobiilipelit ovat videopelejä, jotka on suunniteltu pelattavaksi useimmiten esimerkiksi matkapuhelimella, tablet-tietokoneella ja älypuhelimella. Tästä voisi olettaa, että mobiilipelejä olisivat kaikki kannettavalla laitteella pelattavat pelit, mutta näin ei kuitenkaan ole. Niin sanottujen käsikonsolien, kuten PlayStation Portable tai Nintendo Game Boy, pelit eivät lukeudu mobiilipeleiksi, vaikka ne mukana kätevästi kulkevatkin. [1.]

Mobiilipelien historia alkoi vuodesta 1997, jolloin suomalainen matkapuhelinyhtiö Nokia julkaisi Nokia 6110 -matkapuhelimen. Tähän matkapuhelimeen oli esiasennettu Snake, joka tunnetaan myös nimellä matopeli. Matkapuhelimissa näytöt olivat silloin hyvin pieniä ja kaksivärisiä, joten peli ei ollut mitenkään graafisesti hieno, vaan yksinkertaisia mustista pikseleistä koostuvia muotoja vihreällä taustalla. Pelin tavoite oli niinkin yksinkertainen, että pelaajan täytyi ohjata pelihahmona toimivaa matoa saaliiden luo. Peli loppui jos mato törmäsi seinään tai itseensä. Mato kasvoi jokaisen syödyn saaliin jälkeen, joka toi peliin lisää vaikeusastetta koko ajan, koska madon kasvaessa pelialue käytännössä pieneni. [2.] Madon ohjaaminen tapahtui matkapuhelimen numeronäppäimistöä hyödyntäen, mikä ei sinäänsä ollut kovin käyttäjäystävällinen, sillä näppäimistöä ei oltu suunniteltu pelaamiseen.

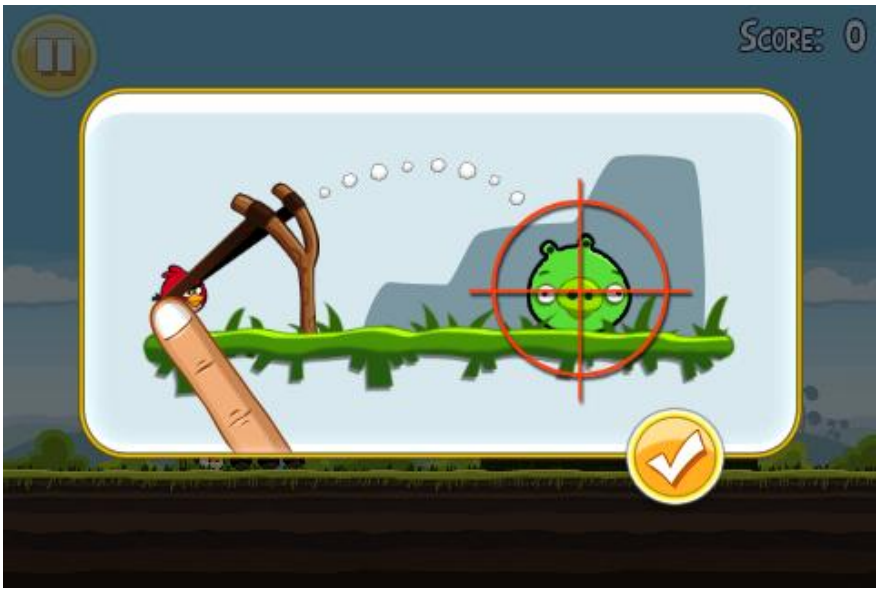


Kuva 1 Pelikuvaa matopelistä

2000-luvun alkupuolella kuluttajamarkkinoille alkoi saapua värinäytöllisiä matkapuhelimia, joten mobiilipelaaminen koki uuden merkittävän käänteen, kun pelit saivat syvyyttä värien kautta. Toinen merkittävä asia, joka kiihdytti mobiilipelien kehittymistä oli Java 2 Micro Edition (J2ME) ja se, että uudet laitteet alkoivat tukea sitä. [2]. J2ME on sulautetuille järjestelmille tarkoitettu versio Javasta. J2ME mahdollisti sen, että samaa koodia pystyi ajamaan useilla eri laitteilla ja jopa eri valmistajien laitteilla. Pelejä ja ohjelmia pystyi myös kehittämään ja testaamaan työpöytäympäristössä, joka nopeutti kehitystyötä. [3.]

Kuva 2 Pelikuvaa Jamdat Bowling pelistä

Vuonna 2007 alkoi matkapuhelimien uusi aikakausi ja tämän myötä myös uusi aikakausi mobiilipeleille, kun yhdysvaltalainen yritys Apple julkaisi sen ensimmäisen matkapuhelimien iPhonen. iPhone oli kosketusnäytöllinen laite, jonka käyttöjärjestelmä oli optimoitu täysin käytettäväksi pelkästään laitteen kosketusnäyttöä hyödyntäen. [4.] Tämä tarkoitti mobiilipelien kannalta täysin uudenlaista tapaa kontrolloida peliä. Hyvänä esimerkkinä tästä uudesta tavasta pelata on Rovion vuonna 2009 julkaisema Angry Birds -peli, jossa yksinkertaisuudessaan oli tarkoituksena lingota ritsalla lintuja possuja päin. Peli käytti hyödyksi kosketusnäyttöä siten, että ritsalla olevasta linnusta ”otettiin kiinni” painamalla sitä sormella, jonka jälkeen sormea vieritettiin laitteen ruudulla taaksepäin ja päästettiin irti. Tämä tapa lingota lintu jäljitteli hyvin tapaa, miten oikeassakin maailmassa henkilö ampuisi ritsalla.



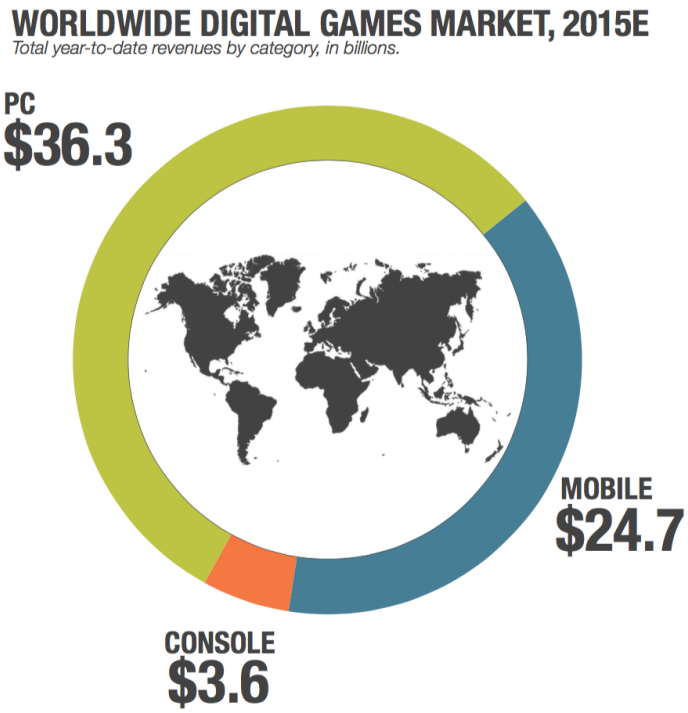
Kuva 3. Angry Birds -pelin ohjekuva

Nykyään mobiilipelit ovat kehittyneet niin pitkälle, että ne muistuttavat jo erittäin paljon konsolipelejä. Tästä hyvänä esimerkkinä on Gameloftin julkaisema Modern Combat -pelisarja, jonka pelit ovat 3D FPS -pelejä. Pelissä pelaaja toimii sotilaana, joka suorittaa määrättyjä tehtäviä tuhoamalla kohteita käyttäen erilaisia aseita. Uusimmissa sarjan peleissä pelaaja voi pelata verkon yli toisia pelaajia vastaan. Pelisarjaa on julkaistu viisi eri versiota mobiililaitteille. Pelisarjassa on erittäin paljon vaikutteita hyvin kuuluisista konsoleille ja PC:lle julkaistuista pelisarjoista, kuten Call of duty ja Battlefield.[5.]

Kuva 4 Modern Combat 5: blackout

3 Mobiilipelikehitys

Mobiilipelit ovat nousseet yhdeksi tärkeimmistä alustoista niin pelaajille kuin myös pelikehittäjille ja julkaisijoille. Vuonna 2015 mobiilipelien maailmanlaajuinen tuotto oli noin 25 miljardia dollaria. Vertailukohteeksi voidaan ottaa toinen erittäin pitkään markkinoilla ollut alusta eli PC, jonka maailmanlaajuinen tuotto oli samana vuonna noin 36 miljardia dollaria. Ottaen huomioon sen, että mobiilipelit on vielä nuori alusta, on se silti kasvanut pienessä ajassa isoksi tekijäksi maailmanlaajuisesti.[6.]



Kuva 5 Maailmanlaajuinen tuottojakauma eri alustojen välillä

Indie-kehittäjille mobiilipelit ovat helpoin tapa saada oma pelinsä kaikkien nähtäväksi, sillä Applen App storeen ja Googlen Play -sovelluskauppaan voi kuka tahansa julkaista omia pelejään. Kehittäjä voi tehdä peleillään rahaa lisäämällä peleihin esimerkiksi mainoksia tai pelin sisäisiä mikromaksuja, joilla pelaaja voi ostaa itselleen oikealla rahalla peliin sisältöä esimerkiksi hahmolleen uusia asusteita.

3.1 Työkaluja mobiilipelikehitykseen

Kehittäjille on tarjolla monenlaisia työkaluja, pelimoottoreita ja sovelluskehyksiä pelien tekemiseen, joista kukin voi valita mieleisensä. Kun mobiilipeliä lähdetään tekemään, on hyvä päättää projektin alussa esimerkiksi, että millä ohjelmointikielellä projekti halutaan toteuttaa ja mille mobiilialustoille. Seuraavaksi esitellään muutama tunnettu pelinkehityssovelluskehys.

3.1.1 Unity

Unity on ehkä jopa eniten käytetty sovelluskehys mobiilipeleille. Esimerkiksi hyvin moni iOS-peli on tehty käyttäen unityä. Unity tukee kolmea eri ohjelmointikieltä, jotka ovat C#, UnityScript ja Boo. Unity on järjestelmäriippumaton sovelluskehys, joka tarkoittaa sitä, että sillä tehtyjä pelejä voidaan ajaa joko mobiilissa, PC:llä tai tunnetuilla konsoleilla kuten Xbox ja PS3. Unity tarjoaa kaksi versiota: ilmaisversiolla käyttäjä voi kehittää pelejä mobiiliin ja PC:lle, kun taas maksullinen versio tarjoaa myös tuen konsolikehitykselle.[7.]

Unity on hyvin monipuolinen ja monenlaiseen projektiin soveltuva sovelluskehys. Sillä voidaan tehdä yksinkertaisista 2D-tasohyppelypeleistä aina vaativiin monipelattaviin 3D FPS -peleihin asti ja sillä kehittäminen on myös erittäin nopeaa. Tästä syystä unity on niin suosittu kehittäjien keskuudessa.

3.1.2 Cocos2D-X

Cocos2D-X on avoimeen lähdekoodiin perustuva järjestelmäriippumaton sovelluskehys, jolla voidaan tehdä näyttäviä 2D-pelejä mobiilille ja PC:lle. Cocos2D-X tukee kolmea eri ohjelmointikieltä C++:aa, Luaa ja JavaScriptiä. Cocos-2D-X sisältää monia työkaluja, jotka helpottavat pelien tekemistä. Muutamia näistä työkaluista ovat esimerkiksi sisäänrakennettu fysiikkamoottori, käyttöliittymätyökalut, äänentoisto ja verkkotyökalut. [8.]

Cocos2D-X:ää käyttävät monet isot tekijät mobiilipelialalla. Muun muassa Zynga, Glu ja suomalainen mobiilipeliyhtiö Fingersoft käyttävät Cocos2D-X:ää peliensä tuottamiseen. [8.]

4 Ohjelmointikielet

Projektin ohjelmointikieleksi on valittu Java pääsääntöisesti siksi, koska projektin toteutuksessa käytettävä sovelluskehys LibGDX käyttää ohjelmointikielenä Javaa. Myös projektin tiimin sisällä on aikaisempaa kokemusta javalla ohjelmoimisesta, joten se helpottaa projektin etenemistä huomattavasti. Seuraavaksi kuitenkin esitellään javan ohella lyhyesti myös C++ ja C#.

4.1 C++

C++ on näistä kolmesta kielestä vanhin. Sen kehitti Bjarne Stroustrup vuonna 1979. C++ on käytännössä C-ohjelmointikielen lisäosa. C++ tukee proseduraalista, olio-ohjelmointia ja geneeristä ohjelmointia. Kun vertailee C++ ja javan syntakseja, niin huomaa väkisinkin yhtenäisyyksiä, sillä java on ottanut erittäin paljon vaikutteita C++:sta. Eroavaisuuksia kuitenkin löytyy erittäin paljon. Suurin ero on se, että C++ ohjelmaa ajetaan suoraan laitteen raudalla, kun taas java-ohjelmaa ajetaan virtuaaliympäristössä. Tämän takia C++-ohjelmat ovat yleensä paljon suorituskyvyltään paljon parempia. Toinen merkittävä eroavaisuus liittyy muistinhallintaan. C++:ssa kehittäjä pystyy itse määrittämään, milloin esimerkiksi jokin olio poistetaan ja sen varaama muisti vapautetaan uudelleenkäytettäväksi. Javassa niin sanottu roskien keräys on toteutettu automaattisesti, joten kehittäjän ei siitä tarvitse huolehtia. Koska C++:ssa kehittäjä pystyy itse määrittämään, milloin muistia vapautetaan, tulee ohjelmista tehokaampia, mutta se on työläämpää, koska se pitää muistaa ottaa huomioon.

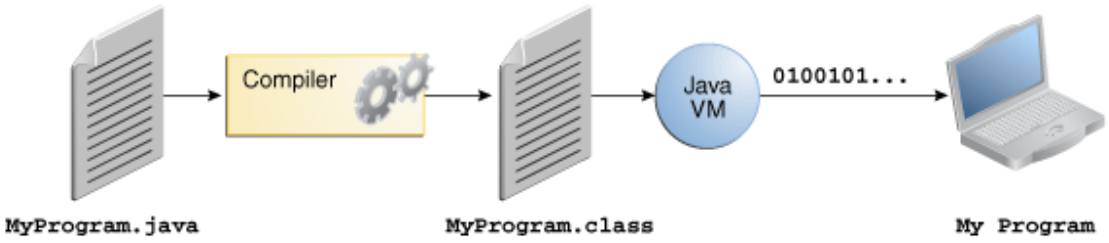
4.2 C#

C# on Microsoftin luoma olio-ohjelmointikieli. C# on hyvin saman tyylinen ohjelmointikieli kuin java niin syntaksiltaan kuin toiminnaltaan. Eroja kuitenkin löytyy, esimerkiksi omien arvotyyppien luominen ei onnistu javassa, kun taas C# antaa käyttäjän luoda omia arvotyyppejä. C#:sta taas puuttuu anonyymit luokan sisäiset luokat, mitä javassa pystyy tekemään. Javasta siirtyminen C#:iin ja päinvastoin on kuitenkin yleensä hyvin helppoa.

4.3 Java

Java on Sun Microsystemsin vuonna 1995 julkaisema alusta ja ohjelmointikieli, joka on nykyään Oraclen hallinnoima ja ylläpitämä. Java on globaali standardi sulautetuille sovelluksille, mobiilisovelluksille, peleille, verkkosisällölle ja yritysohjelmistoille. Yli 9 miljoonaa sovelluskehittäjää maailmassa käyttää javaa. [9.]

Java on korkean tason ohjelmointikieli. Java-ohjelmia ajetaan java virtuaalikoneella (Java Virtual Machine JVM). Java-ohjelmoinnissa kaikki lähdekoodi kirjoitetaan puhtaana tekstinä, mikä on ihmisille täysin luettavaa. Tämä tekee siitä korkean tason ohjelmointikielen. Lähdekoodin tiedostot loppuvat päättellä *.java*, jotka sitten ohjelman koontivaiheessa kootaan *.class*-tiedostoiksi. Nämä *.class*-tiedostot sisältävät tavukoodia, joka on JVM:n kieli. [10.]



Kuva 6 Java-ohjelman koontiprosessin havainnollistaminen [10.]

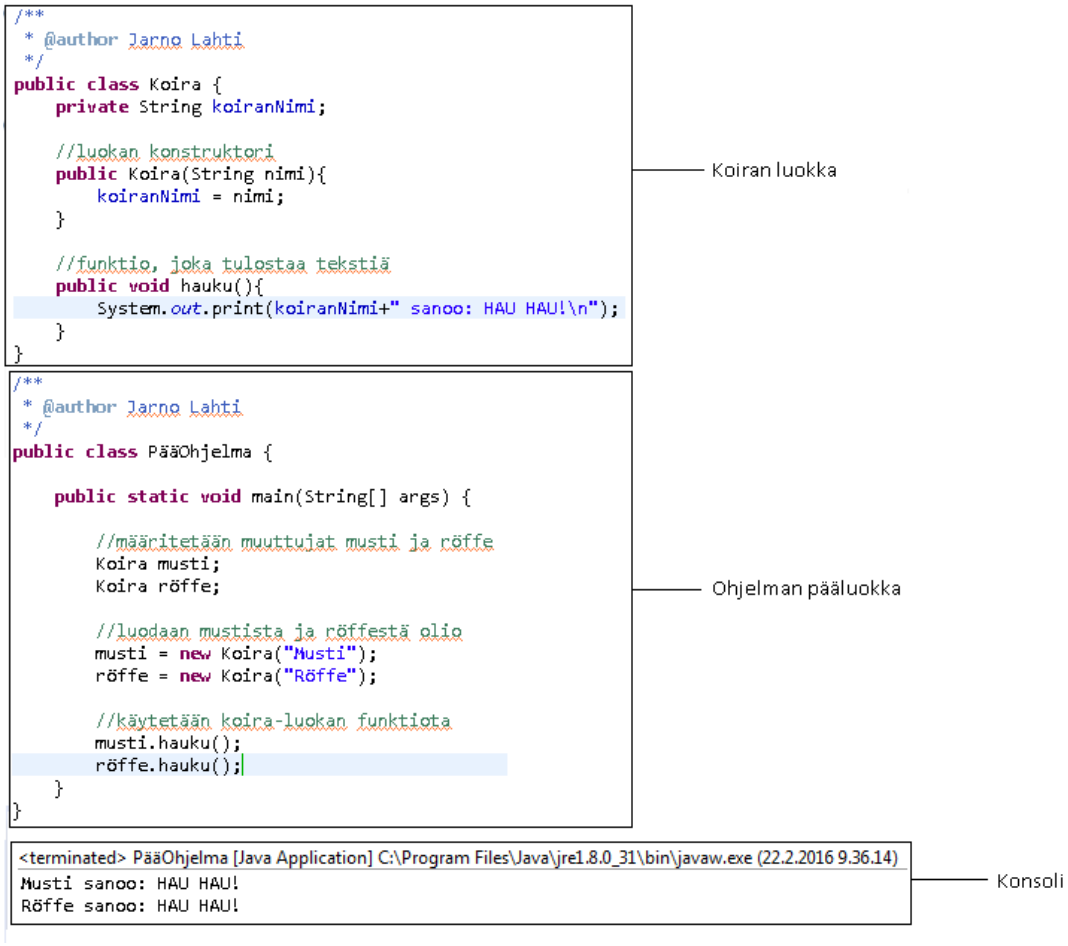
Koska JVM on saatavilla monelle eri käyttöjärjestelmälle, tarkoittaa tämä sitä, että samaa koodia voidaan ajaa millä tahansa laitteella, joka tukee JVM:ää. Koska ohjelmia ajetaan virtuaaliympäristössä, voi ohjelmien suorituskyky olla hieman heikompi kuin laitteen natiivilla ohjelmointikielellä kirjoitettu ohjelma. [10.]

5 Olio-ohjelmointi

Olio-ohjelmoinnilla viitataan ohjelmointityyppiin, jossa ohjelmoija luo datarakenteeseen myös toiminnot eli funktiot, joita tämä datarakenne voi käyttää. Tällä tavalla datarakenteesta tulee olio, joka sisältää tiedon lisäksi myös toimintoja. Tärkeimpiä etuja olio-ohjelmoinnille verrattuna proseduraaliseen ohjelmointiin on se, että ohjelmoija voi luoda moduuleja, joita ei tarvitse muuttaa, kun halutaan luoda uudentyyppinen objekti. Uusi objekti voi periä toiminnallisuutta jo toimivalta vanhalta objektilta, mikä tekee olio-ohjelmointia hyödyntävien ohjelmien muokkaamisesta helppoa. [11.]

Java on luokkapohjainen olio-ohjelmointikieli. Luokkapohjaisessa olio-ohjelmoinnissa keskeisenä asiana ovat luokat. Luokat tarjoavat tavan määritellä joukon olioita ja toiminnallisuuden manipuloida niitä. Oliot ovat ilmentymiä luokista. Luokat ovat niin sanotusti muotteja jostain asiasta. Luokka sisältää datan ja operaatiot, joita datalle voidaan suorittaa. Luokka voi esimerkiksi kuvata linkattua listaa, tietokoneen näytöllä näkyvää

ikkunaa, tiedostoa tai vaikkapa huonekalua. [12, s.16]. Seuraavassa esimerkkikoodissa havainnollistetaan, miltä luokka voi näyttää sisältäpäin ja miten sitä käytetään pääohjelmassa. Esimerkiksi on otettu koira, josta tehdään luokka ja esitellään, kuinka tätä luokkaa käytetään.



Esimerkkikoodi 1 Luokan rakenteen, olioiden luomisen ja käyttämisen havainnollistaminen

Esimerkkikoodissa 1 on ensin esitettynä Koira -luokka, joka sisältää String-tyyppisen muuttujan *koiranNimi*, luokan konstruktori, joka ottaa vastaan String tyyppisen muuttujan *nimi*, joka sitten asetetaan edellä mainittuun *koiranNimi*-muuttujaan. Viimeiseksi luokasta löytyy funktio, joka tulostaa tekstin konsoliin. Seuraavaksi on esitelty ohjelman pääluokka, jossa on ensin määritettynä kaksi Koira-tyyppistä muuttujaa *musti* ja *röffe.* Seuraavaksi näistä kahdesta muuttujasta luodaan oliot kutsumalla Koira-luokan konstruktoria, joka ottaa vastaan nimen String-tyyppisenä muuttujana. Viimeiseksi mo-

lemmat oliot kutsuvat Koira-luokan funktiota *hauku,* jonka jälkeen konsoliin tulostuu tekstiä.

6 Insinöörityössä käytetyt työkalut

6.1 Android studio

Android Studio on virallinen ohjelmointiympäristö Android-applikaatioille, ja se perustuu IntelliJ IDEA Java -ohjelmointiympäristöön. Se tarjoaa kehittäjälle useita työkaluja, kuten esimerkiksi:

- gradleen perustuvan koontijärjestelmän
- koontivariaatioita ja moni-APK tiedosto generaation
- valmiita koodipohjia, jotka auttavat yleisten toimintojen tekemisessä
- sommitteluedi torin, jossa on tuettuna raahaa ja pudota –teemaeditointi
- lint –työkaluja joiden avulla voidaan napata suorituskykyyn, käytettävyyteen, yhteensopivuuteen ja muihin ongelmiin liittyviä virheitä
- proGuard ja applikaation signeeraus
- sisäänrakennettu tuki googlen pilvipalveluille.

Android-studio tarjoaa myös monia muita hyödyllisiä työkaluja. [13.]

6.2 LibGDX-viitekehys

Libdgx on avoimeen lähdekoodiin perustuva järjestelmäriippumaton peli- ja applikaatioviitekehys, joka tukee Windows-, Linux-, Mac OS X-, Android-, Blackberry-, iOS- ja HTML5-alustoja. Libgdx mahdollistaa sen, että kehittäjä voi kirjoittaa koodia, ja se kääntyy kaikille yllä mainituille alustoille ilman minkäänlaisia muutoksia koodiin. Tämä

mahdollistaa nopean pelin kehityksen, koska kehittäjä voi käytännössä ohjelmoida ja testata peliä, joka on tarkoitettu pelattavaksi esimerkiksi mobiililaitteella, täysin työpöytäympäristössä. Libgdx käyttää Javaa ohjelmointikielenä, ja se antaa kehittäjälle käyttöön koko Javan tarjoaman ekosysteemin, mikä mahdollistaa kehittäjän olla luova. Libgdx tähtää siihen, että se ei olisi pelimoottori vaan lähempänä viitekehystä. Se antaa kehittäjälle vahvat työkalut, joista valita ja antaa kehittäjän itse päättää, miten haluaa pelin tai applikaationsa kirjoittaa. [14.]

Liittessä 1 on esitettynä Android- ja työpöytäsovelluksen käynnistysluokat. Nämä luokat sisältävät mahdolliset alustariippuvaiset alustukset ja asetukset. Esimerkiksi luokassa DesktopLauncher alustetaan ensimmäisenä olio config, joka on tyyppiä LwjglApplicationConfiguration. Tälle oliolle voitaisiin sitten määrittää esimerkiksi applikaation ruudun leveys ja korkeus pikseleissä tai vaikka asettaa applikaatio kokonäyttötilaan. Kun mahdolliset asetukset on määritetty, siirrytään itse applikaation avaamiseen. Työpöytäsovelluksella luodaan uusi LwjglApplication-tyyppiä oleva instanssi, jolle annetaan parametreiksi edellä mainitty config-olio ja uusi instanssi projektin pääluokasta, jossa sijaitsee applikaation toimintalogiikka. Tämä olio sitten käynnistää applikaation käyttäjälle näkyvän osuuden eli tässä tapauksessa peliruudun, joka on esimerkiksi leveydeltään ja korkeudeltaan edellä mainitun config-olion määritysten mukainen.

Libgdx on avoin viitekehys, sillä se antaa kehittäjän käyttää alhaisen tason toimintoja, kuten tiedostojärjestelmiä, syöttölaitteita, audiolaitteita sekä OpenGL:ää yhdistetyn OpenGL ES 2.0 ja 3.0 rajapinnan kautta. Näiden alhaisen tason laitteiden päälle on rakennettu useita sovellusrajapintoja, joiden avulla yleiset toiminnot, kuten esimerkiksi spritejen ja tekstin renderöinti, käyttöliittymien rakentaminen, musiikin toistaminen, erilaisten matemaattisten funktioiden suorittaminen ja eri tietotyyppien jäsentely, onnistuu vaivatta. [14.]

Libgdx tekee tarvittaessa kaikki natiiviin koodiin liittyvät toiminnot itsenäisesti, ja ne suoritetaan yleensä koodin käännösvaiheessa, jolloin kehittäjän ei tästä tarvitse välittää [14]. Natiivilla koodilla tarkoitetaan koodia, joka on suunniteltu ajettavaksi määrätynlaisella prosessorityypillä. Natiivia koodia ei siis pysty ajamaan muilla prosessoreilla ellei koodia ole tarkoitettu emuloitavaksi. [15.]

Libgdx-sovelluksen elinkaari koostuu viidestä eri vaiheesta, jotka ovat luonti (create), pysäytys (pause), jatkaminen (resume), renderöinti (render) ja hävittäminen (dispose).

Näihin ohjelman vaiheisiin kehittäjä pääsee käsiksi implementoimalla applikaationsa pääluokkaan libgdx:n oman rajapinnan *ApplicationListener*. ApplicationListener -rajapinta sisältää metodit create, resize, render, pause, resume ja dispose, joiden avulla kehittäjä pääsee käsiksi sovelluksen elinkaareen.[16.] Metodien käyttötarkoitukset ovat seuraavat:

- **Create:** Tätä metodia kutsutaan vain kerran sovelluksen käynnistyksen yhteydessä, joten siinä voi tehdä esimerkiksi sovellukseen liittyvät initialisaatiot

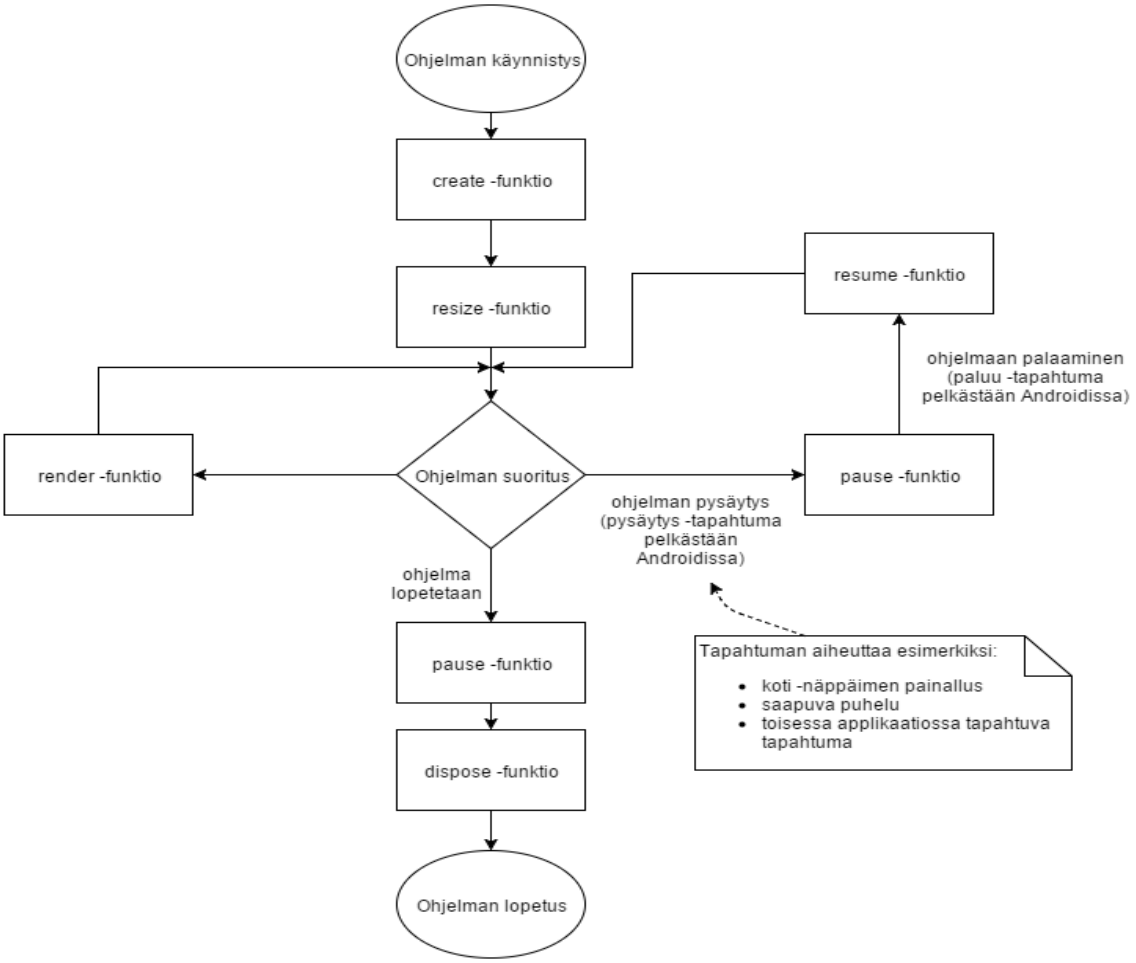
- **Resize:** Tätä metodia kutsutaan heti createn jälkeen ja myös aina kun ohjelman ikkunana kokoa muutetaan esimerkiksi työpöytäsovelluksessa. Tämä metodi ottaa vastaan kaksi int-tyyppistä parametriä, jotka ovat pikselileveys ja -korkeus.

- **Render:** Tätä metodia kutsutaan sovelluksen pääsilmukassa aina, kun piirtämistä kuuluu tapahtua. Tätä metodia käytetään yleensä myös esimerkiksi pelilogiikan päivittämiseen.

- **Pause:** Kaikilla sovellusalustoilla tätä metodia kutsutaan, kun sovellusta ollaan lopettamassa. Jos sovellusalustana on Android, niin tätä metodia kutsutaan, kun sovellus halutaan pysäyttää. Tällaisia tilanteita ovat esimerkiksi, kun käyttäjälle tulee puhelu tai käyttäjä painaa laitteen koti-näppäintä.

- **Resume:** Tätä metodia kutsutaan pelkästään Androidilla. Kutsu tapahtuu pause-metodin jälkeen, kun sovellukseen ollaan palaamassa esimerkiksi, kun käyttäjä lopettaa puhelun, joka keskeytti sovelluksen aikaisemmin.

- **Dispose**: Tätä metodia kutsutaan kun sovellus ollaan lopettamassa. Ennen dispose -metodin kutsumista kutsutaan pause-metodia.



Kuva 3 Libgdx-sovelluksen elinkaari suomeksi havainnollistettuna [16].

Libgdx:n ydin koostuu kuudesta eri moduulista, jotka tarjoavat keinon kommunikoida käyttöjärjestelmän kanssa. Näitä moduuleja voidaan kutsua kaikissa järjestelmissä, ja ne toimivat kaikissa samalla tavalla. [17.] Moduulit ovat seuraavat:

- **Sovellus(Application):** Suorittaa applikaation ja pitää huolen siitä, että API on tietoinen applikaation tason tapahtumista, kuten esimerkiksi ikkunan koon muutoksista.

- **Tiedostot(Files):** Mahdollistaa tiedostojärjestelmän käytön eri alustoilla.

- **Syöttö(Input):** Informoi API:a käyttäjän syöttämistä komennoista, kuten hiiren klikkauksista ja näppäimistön tai kosketusnäytön painalluksista.

- **Verkko (Net):** Tarjoaa tavat päästä käsiksi resursseihin HTTP- tai HTTPS-protokollia hyödyntäen.

- **Ääni (Audio):** Tarjoaa tavat toisaa ääniefektejä ja musiikkia. Mahdollistaa myös pääsemisen äänilaiteisiin käsiksi koodissa.

- **Grafiikka (Graphics):** Mahdollistaa käsiksi pääsemisen OpenGL ES 2.0 –rajapintaan.

Näihin moduuleihin pääsee käsiksi koodissa Gdx-luokan kautta, jossa ne ovat staattisina muuttujina. [17.] Seuraavassa esimerkkikoodissa havainnollistan, kuinka näihin moduuleihin pääsee käsiksi koodissa.

```
/**
 *  Kaikki moduulit löytyvät staattisina referensseinä luokasta Gdx, josta niihin pääsee käsiksi
 *  missä tahansa.
 */

//Alustaa Application -moduulin muuttujaan _application joka on tyyppiä Application.
Application _application = Gdx.app;

//Alustaa Files -moduulin muuttujaan _files joka on tyyppiä Files
Files _files = Gdx.files;

//Alustaa Input -moduulin muuttujaan _input joka on tyyppiä Input
Input _input = Gdx.input;

//Alustaa Net -moduulin muuttujaan _net joka on tyyppiä Net
Net _net = Gdx.net;

//Alustaa Audio -moduulin muuttujaan _audio joka on tyyppiä Audio
Audio _audio = Gdx.audio;

//Alustaa Graphics -moduulin muuttujaan _graphics joka on tyyppiä Graphics
Graphics _graphics = Gdx.graphics;
```

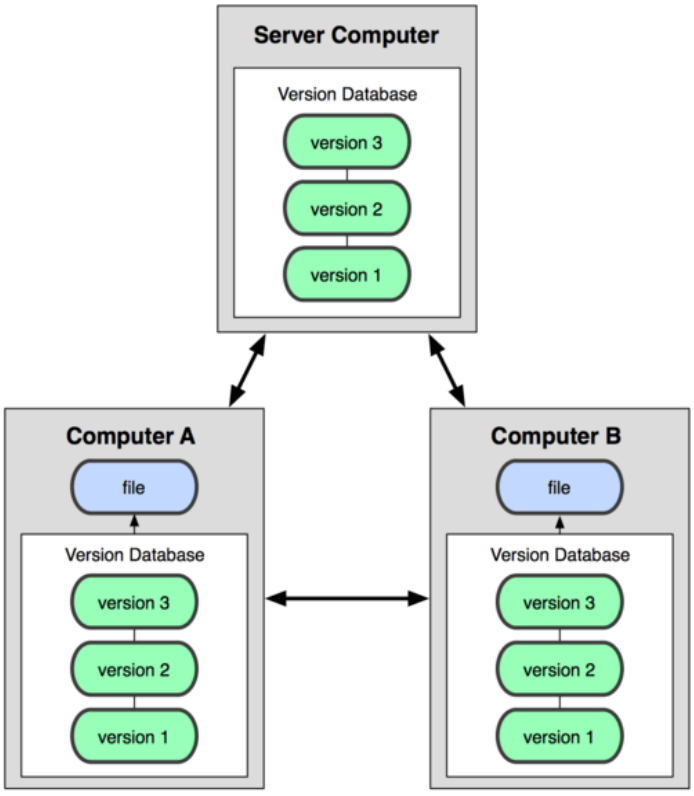
Esimerkkikoodi 2 Moduulien alustus

6.3 Git-versionhallintajärjestelmä

Versionhallinta on tärkeä osa-alue ohjelmistokehityksessä, koska sillä helpotetaan tiimityöskentelyn toimintaa huomattavasti. Versionhallinnalla tarkoitetaan yleensä työkalua tai menetelmää, jolla pidetään projektissa olevat tiedostot ajan tasalla, ja virhetilan-

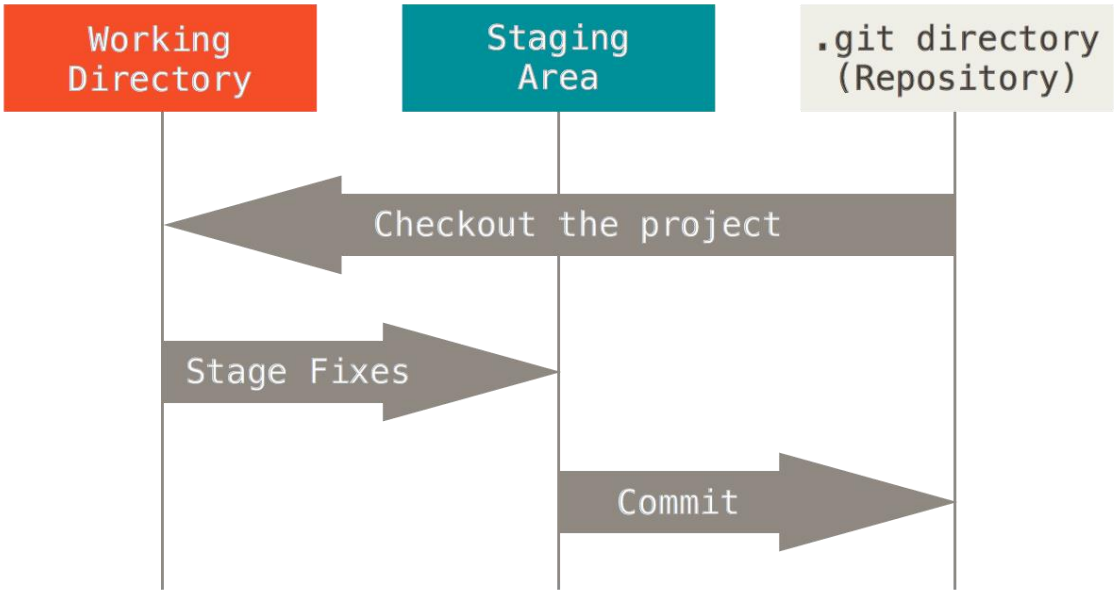
teiden sattuessa se mahdollistaa myös paluun edelliseen versioon. Pienissä ohjelmistoprojekteissa versionhallinta voidaan toteuttaa jopa niin yksinkertaisesti, että tiedostoja kopioidaan paikallisesti vain uusiin kansioihin talteen ja esimerkiksi nimetään kansio versionumerolla. Tämä toimii tiettyyn pisteeseen saakka, mutta on erittäin herkkä ongelmatilanteille, kuten esimerkiksi kovalevyn rikkoutumiselle tai muuten vaan datan katoamiselle. [18, s.1]

Git on versionhallintajärjestelmä, joka perustuu hajautettuun versionhallintamalliin. Hajautetulla versionhallinnalla tarkoitetaan sitä, että projektin tiedostoilla on jokin keskitetty tietokanta, josta jokainen käyttäjä kopioi koko tietokannan paikallisesti omalle työkoneelleen. Käyttäjät tekevät muutoksia näihin paikallisiin tiedostoihin ja lopulta lähettävät itse tekemät muutokset keskitettyyn tietokantaan. Koska koko tietokannan sisältö on jokaisen omalla työkoneella ja kaikki muutokset tehdään paikallistiedostoihin, mahdollistaa tämä helpon tavan palauttaa varmuuskopio tietokannasta, jos esimerkiksi palvelin hajoaa. [18, s.4]



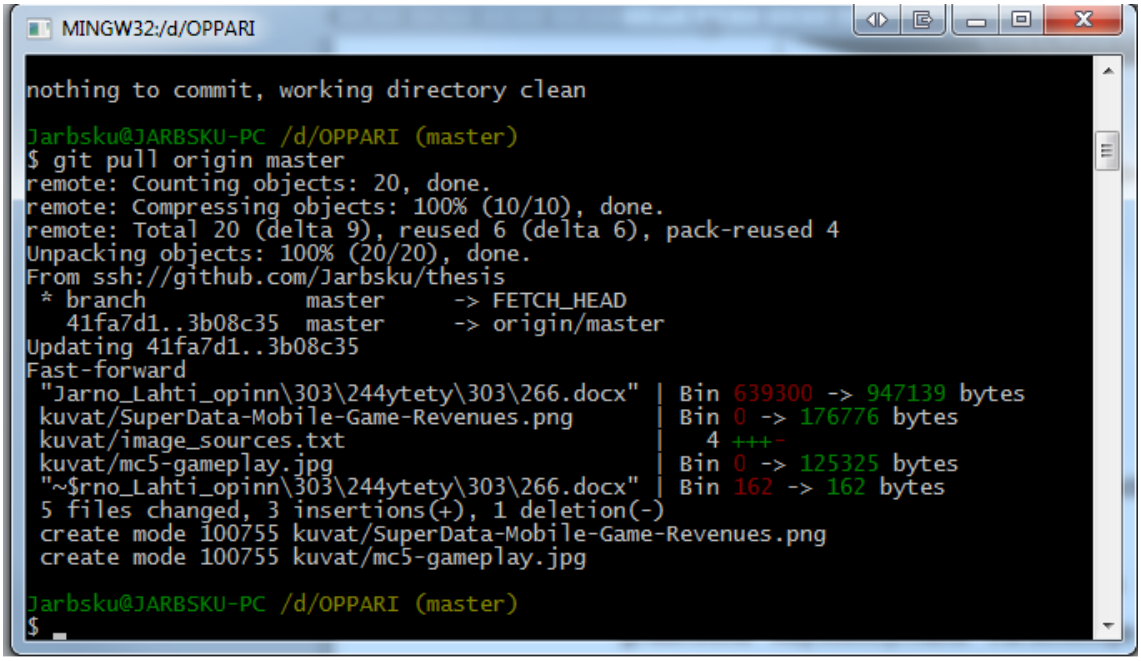
Kuva 4. Hajautetun versionhallinnan havainnollistamiskaavio.

Git: n toiminta perustuu kolmeen eri vaiheeseen, joissa tiedostot voivat olla. Kommitoituna (commited), modifioituina (modified) tai esitetty (staged). Kommitoitu tarkoittaa, että data on tallennettu paikalliseen tietokantaan. Modifioitu tarkoittaa, että tiedostoissa on tapahtunut muutoksia, mutta niitä ei ole vielä kommitoitu. Esitetty tarkoittaa sitä, että muokatuista tiedostoista on otettu talteen nykyinen tila ja määritetty se menemään seuraavassa tallennuksessa paikalliseen tietokantaan. Tämä tarkoittaa sitä, että git-projektissa on kolme eri lohkoa, joissa tiedostot ovat Git-kansio, työkansio ja esitysalue. Git-kansio sisältää metadatan ja projektin tietokannan. Työkansio sisältää kopiot tietokannan tiedostoista, joita käyttäjä muokkaa. Esitysalue on git-kansion sisällä oleva tiedosto, joka sisältää informaatiota seuraavaan kommittiin menevästä datasta. [18, s.8]



Kuva 5. Gitin kolme päätilaa ja niissä tiedostojen liikkuminen.

Työskentely giti:llä tapahtuu yleensä komentorivin kautta, mutta vaihtoehtoisia graafisella käyttöliittymällä varustettuja ohjelmistoja on myös saatavilla. Näissä graafisissa ohjelmistoissa on yleensä ongelmana se, että niillä ei voi tehdä kaikkea mahdollista, mitä komentorivillä työskentelyllä voi tehdä. Tämä johtuu siitä, että graafiset ohjelmistot usein tarjoavat käyttäjälle vain osan suoritettavista komennoista. [18, s.9.]



Kuva 7 Gitin komentorivityökalu

7 SumTower

SumTower on matemaattinen pulmapeli, missä pelaajan on tarkoitus saada muodostettua pelilaudalle mahdollisimman suuri yhtenäinen summa. Pelilaudalla on yhteensä 24 palikkaa 4x6-ruudukossa, jotka toimivat peliobjekteina. Palikka sisältää positiivisen luvun, joka on alussa 1. Palikka voi olla yksi seitsemästä eri väristä, jotka ovat seuraavat: vihreä, punainen, turkoosi, keltainen, oranssi, sininen ja lila. Pelaaja poistaa palikoita koskettamalla peliruutua halutun palikan kohdalta, jonka jälkeen palikka ja sen sisältämä luku häviävät pelilaudalta. Tämän jälkeen poistetun palikan yläpuolella ollut palikka tippuu alas ja ”syö” mahdolliset viereiset sekä alapuolella olevan samanvärisen palikan ja lisää omaan lukuunsa kaikkien syötyjen palikoiden sisältämät luvut. Tätä kutsutaan pelissä mergeämiseksi. Uusia palikoita ilmestyy pelilaudan yläriville aina, kun siellä on tilaa. Ilmestyvillä palikoilla on satunnaisesti generoitu väri seitsemästä ylempänä mainitusta väristä ja lukuarvona yksi.

Kuva 8 SumTower pelilauta pelin alussa

7.1 Pelin tavoite

Tavoite pelissä on saada mahdollisimman iso summa pelilaudalle poistamalla palikoita, jotka takaavat pelaajalle hyvän siirron. Esimerkiksi pelin alussa kaikki 24 palikkaa sisältävät numeron yksi, jolloin pelilaudan kokonaissumma on 24. Kun pelaaja poistaa palikan laudalta, joka esimerkiksi takaa hänelle kahden samanvärisen palikan mergeämisen. Tällöin mergeävä palikka saa luvukseen kaksi ja lauta täyttyy automaattisesti takaisin täyteen, jonka jälkeen laudan kokonaussumma on 25. Tätä sääntöä käyttäen pelaaja sitten etenee pelissä ja yrittää saada mahdollisimman ison summan laudalleen.

Kuva 9 SumTower pelilauta muutaman siirron jälkeen

Pelilaudalle voi pelin aikana tulla tilanteita, missä pelaaja huomaa, että hänellä on mahdollista tehdä vain huono siirto. Tähän ongelmaan pelaajalla on käytössään työkaluja, joita pelissä kutsutaan boostereiksi.

7.2 Palkitseminen ja virhesiirrot

Chain: Kun pelaaja tekee siirron, jonka johdosta pelilaudalla tapahtuu useampi merge, palkitaan pelaajaa tälläisesta onnistuneesta siirrosta siten, että jokaiseen chainissa mukana olleeseen palikkaan lisätään siirron jälkeen vielä yksi kyseisen palikan sisältämään lukuun.

Double merge: Kun pelaaja tekee siirron, jonka johdosta yksi palikka mergeää kaksi samanväristä palikkaa ympäriltään, pelaajaa palkitaan tällaisesta siirrosta. Palkinto riippuu, missä pelimuodossa pelaaja siirron tekee.

Triple merge: Kun pelaaja tekee siirron, jonka johdosta yksi palikka mergeää kolme samanväristä palikkaa ympäriltään. Pelaajaa palkitaan tälläisestä siirrosta. Palkinto riippuu, missä pelimuodossa pelaaja siirron tekee.

Virhesiirto: Kun pelaaja tekee siirron, jonka johdosta yksikään palikka ei mergeä. Tästä siirrosta, pelaajaa rangaistaan siten, että viidestä satunnaisesta pelilaudan palikasta vähennetään yksi. Palikoiden sisältämä luku ei kuitenkaan voi mennä alle nollan.

7.3 Pelimuodot

SumTower sisältää kolme eri pelimuotoa. Pelimuodot ovat turn, time ja journey mode. Jokainen näistä muodoista sisältää omat sääntönsä ja jokaisessa on oma uniikki palkitsemistapa ylläesitetyille double ja triple mergeille.

7.3.1 Turn mode

Pelaajalle on annettu aluksi 30 siirtoa. Pelaaja yrittää saada annetuilla siirroilla mahdollisimman suuren loppusumman pelilaudalleen. Yllä mainitut double ja triple merget palkitsevat tässä pelimuodossa pelaajaa lisävuoroilla. Double mergestä pelaaja saa yhden lisävuoron ja triple mergestä kaksi lisävuoroa.

7.3.2 Time mode

Pelaajalla on 45 sekunttia aikaa saada mahdollisimman suuri loppusumma pelilaudalleen. Double ja triple mergeistä pelaaja saa tässä pelimuodossa lisäaikaa. Double merge antaa pelaajalle kolme lisäsekuntia ja triple merge antaa pelaajalle viisi lisäsekuntia peliaikaa.

7.3.3 Journey mode

Journey mode eroaa muista pelimuodoista siten, että se ei lopu koskaan. Tässä pelimuodossa pelaaja yrittää rakentaa systemaattisesti tornia palikoista. Ensin pelaajalle annetaan tehtäväksi saada laudalle palikka jonka arvo on 10. Kun pelaaja saa tehtyä palikan, joka vastaa tai on yli annetun luvun, se siirtyy torniin. Tämän jälkeen pelaaja saa uuden luvun, joka hänen pitää muodostaa yhteen palikkaan. Tarvittava luku nousee aina yhdellä ensiksi 10 sitten 11, 12, 13 jne. Tämä pelimuoto ei lopu koskaan, vaikka pelaaja sulkisi pelin. Pelaaja voi milloin tahansa jatkaa peliä siitä kohtaa, mihin hän viimeksi jäi.

8 Pelin toteutus

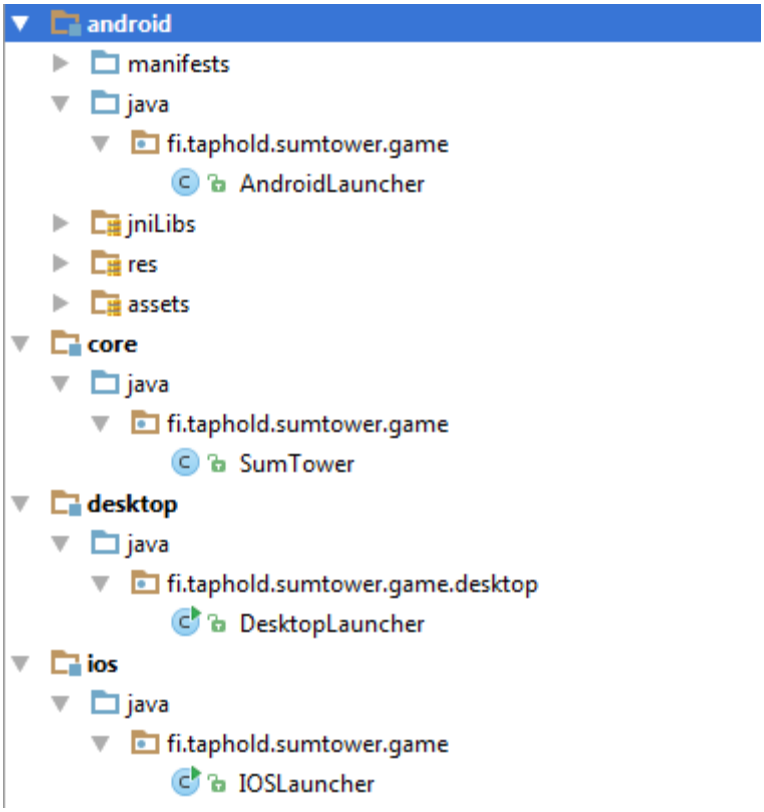
8.1 Projektin luonti

Projekti aloitetaan käyttäen libgdx:n tarjoamaa graafista projekti generaattoria. Generaattoriin syötetään projektin nimi, Java-paketin nimi, pelin pääluokan nimi, projektin sijainti tietokoneella ja viimeiseksi Android SDK:n sijainti tietokoneella. Seuraavaksi valitaan mitkä kaikki alustat halutaan tukea. Vaihtoehtoina ovat Työpöytä, Android, iOS ja HTML5. Lopuksi on mahdollista vielä valita erilaisia lisäosia, mutta nämä ovat valinnaisia.

Kuva 10 Libgdx:n projekti generaattorin käyttöliittymä

Kun projekti on luotu, avataan se kehitysympäristössä. Tässä projektissa kehitysympäristönä toimii Android Studio, mutta koska libgdx käyttää projektin riippuvuuksien ja koontiprosessin hallintaan gradlea, on mahdollista käyttää myös muita kehitysympäristöjä, kuten Eclipseä tai Netbeansiä. Android Studio tukee gradle-projekteja, mutta esim Eclipseen saattaa joutua lataamaan erillisen lisäosan, jotta sillä pystytään kehittämään libgdx-sovelluksia.

Libgdx-projekti koostuu useasta java-projektista. Jokainen kehittäjän valitsema alusta on oma projektinsa, ja ne sisältävät kaiken tarpeellisen koodin, joka koskee kyseistä alustaa. Ne sisältävät myös alustan pääluokan, joka on havainnollistettu liittessä 1. Itse sovelluksen tai pelin koodi kirjoitetaan core-projektiin. Kaikki grafiikat ja äänet laitetaan Android-projektissa olevaan assets-kansioon, joka sitten linkataan jokaiseen projektiin, jotta ne osaavat hakea tiedostoja sieltä, kun niitä koodissa kysytään.



Kuva 11 Libgdx-projekti Android Studiossa

8.2 Versionhallinta

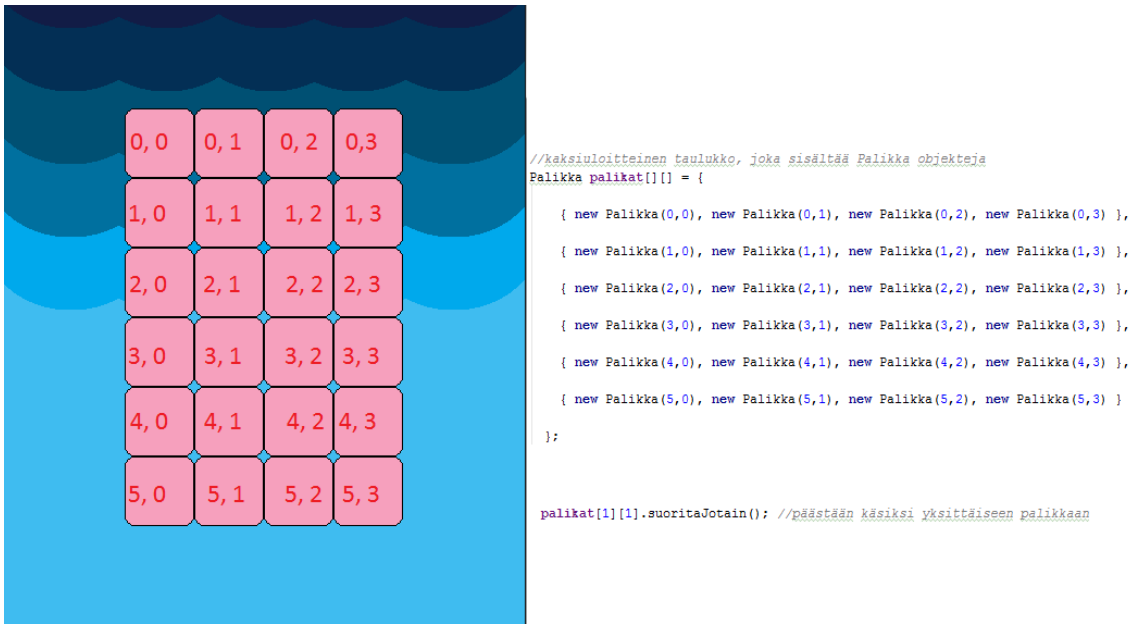
Projektin versionhallintaan käytettiin jo aiemmin esiteltyä Git:iä. Repositorion palveluntarjoajaksi valittiin Bitbucket-niminen palveluntarjoaja. Valinnan syynä oli se, että Bitbucket tajosi mahdollisuuden ilmaiseen yksityiseen repositorioon, johon pääsee käsiksi vain mukaan kutsutut henkilöt. Palvelu on ilmainen niin kauan kuin repositorion käyttöön on kutsuttu maksimissaan viisi henkilöä. Toinen mahdollinen palveluntarjoaja olisi ollut Github, mutta he laskuttavat privaateista repositorioista, joten päätös oli tällä tapaa

hyvin selvä, koska kehittäjätiimi koostui kolmesta henkilöstä, jotka versionhallintaa käyttivät. Repositorion täytyi olla privaatti, ettei kuka tahansa pysty katsastelemaan pelin lähdekoodia.

8.3 Pelimekaniikka

8.3.1 Alustavat määrittelyt

Alustavien määrittelyiden pohjalta lähdettiin rakentamaan peliä. Alussa tarkoitus oli saada järkevästi piirtymään pelilauta ruudun keskelle ja palikoita piti pystyä poistamaan koskemalla niitä sormella. Pelilaudan palikat täytyi sijoittaa jonkinlaiseen taulukkoon, jossa ne säilötään ja josta niitä käsitellään. Mahdollisuuksia oli tehdä joko normaali taulukko, kaksiulotteinen taulukko tai lista, joka sisälsi kaikki laudalla olevat palikka oliot. Palikat päätettiin säilöä kaksiuloitteiseen taulukkoon, koska tällä tavalla oli helppo tietää tasan tarkkaan, että mikä ruudulla näkyvä palikka oli kyseessä, kun sitä haluttiin käsitellä. Seuraava kuva havainnollistaa miltä kaksiuloitteinen taulukko näyttää ja miten taulukon sisältämiä objekteja pystyi käsittelemään.



Kuva 12 Havainnollistaminen palikoiden sijoittelusta kaksiuloitteisessa taulukossa

Yksittäiseen palikkaan päästään käsiksi koodissa erittäin yksinkertaisesti tällä toteutustavalla, ja se on ihmiselle helposti luettava ja ymmärrettävä tapa, joten koodin selkeyden ja luettavuuden vuoksi lähdettiin etenemään kaksiuloitteisella taulukolla.

Palikan luokan rakennetta täyty myös miettiä tässä vaiheessa. Tiedossa oli, että palikka omistaa kokonaisluvun, mitä piti pystyä käsittelemään. Palikalla oli myös väri, mille piti keksiä joku yksinkertainen tapa kertoa se koodissa. Palikan täytyi myös omistaa sprite-tyyppinen olio, jotta palikka pystyttiin piirtämään näytölle. Sprite on libgdx:n oma luokka joka pitää sisällään tiedon tekstuurista, koosta ja sijainnista ruudulla.

8.3.2 Palikka-luokka

Alustava Palikan luokan konstruktori ja luokan sisältämä värin asetus funktio on esiteltynä alla. Konstruktori ottaa vastaan kolme parametriä, joista kaksi ovat Int-tyyppisiä muuttujia, jotka ovat kokonaislukuja. Ensimmäinen on palikan sisältämä luku alussa ja toinen kuvastaa palikan väriä, joka palikalla on. Kolmas parametri, jonka konstruktori ottaa vastaan, on libgdx:n sisältämän Rectangle-tyyppinen olio. Tämä olio pitää sisällään palikan x- ja y-koorditaatit ja myös palikan leveyden ja korkeuden pikseleissä.

```
public Block(int value, int color, Rectangle bounds){
    _bounds = bounds;
    _value = value;

    setColor(color);
    _sprite.setSize(_bounds.width, _bounds.height);
    _sprite.setPosition(_bounds.x, _bounds.y);
}

private void setColor(int color){
    _color = color;

    switch (_color){
        case 0:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_GREEN)));
            break;
        case 1:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_RED)));
            break;
        case 2:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_YELLOW)));
            break;
        case 3:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_BLUE)));
            break;
        case 4:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_MINT)));
            break;
        case 5:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_ORANGE)));
            break;
        case 6:
            _sprite = new Sprite(new Texture(Gdx.files.internal(path+SPRITE_LILA)));
            break;
        default:
            System.out.println("block not set");
            break;
    }
}
```

Esimerkkikoodi 3 Palikan luokan konstruktori ja luokan sisältämä setColor-funktio

Muuttujien nimet, jotka alkavat alaviivalla, viittaavat siihen, että ne ovat luokan omistamia muuttujia. Tämä oli tiimin sisällä sovittu käytäntö, jotta koodia on helpompi lukea. Konstruktorin sisällä asetetaan ensiksi vastaanotettu Rectangle-olio luokan omaan Rectangle-olioon. Seuraavaksi asetetaan palikan lukuarvo, jonka jälkeen konstruktori käyttää luokan sisäistä funktiota setColor. Tässä funktiossa ensiksi asetetaan luokan sisäiseen muuttujaan talteen konstruktorille annettu *color.* Tämän jälkeen luodaan luokan omasta Sprite-tyyppisestä muuttujasta uusi instanssi, jolle asetetaan tekstuuriksi annettua väriä vastaava kuvatiedosto, joka haetaan käyttämällä libgdx:n tarjoamaa moduulia files. Kun setColor-funktio on suoritettu niin lopuksi vielä asetetaan spriten koko ja sijainti käyttäen hyödyksi jo aikaisemmin asetettua *_rectangle* nimistä oliota, josta saadaan koordinaatit x ja y sekä leveys ja korkeus.

8.3.3 Pelilaudan alustus

Jotta palikat voitiin piirtää oikeisiin kohtiin ruudulla, täytyi ne alustaa pelin alussa jo aikaisemmin esitettyyn kaksiuloitteiseen taulukkoon. Ennen palikoiden asettamista täytyi kuitenkin määrittää, minkä kokoinen ja missä kohtaa ruutua pelilauta sijaitsi. Tässä kohtaa täytyi määrittää haluttu virtuaalinen resoluutio pelille. Pelin virtuaaliseksi resoluutioksi valittiin 480 x 800. Jotta lauta saataisiin piirtymään ruudun keskelle, täytyi laskea hiukan matematiikkaa. Tiedettiin jo, että pelilauta koostui 24 palikasta ja ne olivat sijoitettu ruudukkoon, jonka koko oli 4x6. Jotta palikat voitiin alustaa tähän taulukkoon, ensin määrittää, minkä kokoinen pelilaudan tulisi olla pituus- ja leveyssuunnassa. Yhden palikan koko saatiin laskettua jakamalla pelilaudan leveys neljällä ja korkeus kuudella. Palikoiden sijainti ruudulla saatiin määritettyä laskemalla ensin pelilaudan koordinaatit ruudulla, jonka jälkeen palikat voitiin alustaa taulukkoon hyödyntäen näitä tietoja ja käyttäen kahta sisäkkäistä for-silmukkaa. Alla on esitettynä
palikoiden initialisointi taulukkoon.

```
private void initializeBlocks(){
    //calculate width and height of an single block
    float blockWidth = BOARD_WIDTH / BOARD_COLUMNS;
    float blockHeight = BOARD_HEIGHT / BOARD_ROWS;

    //calculate board position to center of the screen
    float boardPositionX = (VIRTUAL_WIDTH / 2) - (BOARD_WIDTH / 2);
    float boardPositionY = (VIRTUAL_HEIGHT / 2) - (BOARD_HEIGHT / 2);

    //initialize blocks to the array
    for(int row = 0; row < BOARD_ROWS; row++){
        for(int col = 0; col < BOARD_COLUMNS; col++){
            float blockPositionX = (blockWidth * col) + boardPositionX;
            float blockPositionY = (blockHeight * row) + boardPositionY;

            blocks[row][col] = new Block(1, randomizeColor(),
                                         new Rectangle(blockWidth,
                                                       blockHeight,
                                                       blockPositionX,
                                                       blockPositionY)
                                        );
        }//end of the inner for-loop
    }//end of the outer for-loop
}
```

Esimerkkikoodi 4 Palikoiden alustus

Aluksi funktiossa lasketaan yksittäisen palikan leveys ja korkeus. Tämän jälkeen lasketaan laudan sijainti ruudulla. Koska peliruutu on 480 pixeliä leveä ja 800 pixeliä korkea, voidaan ruudun keskipiste selvittää helposti jakamalla nämä molemmat luvut kahdella. Ruudun keskipiste on siis kohdassa 240x400. Jotta lauta saataisiin ruudun keskelle, täytyy laudan keskipiste selvittää samalla laskutavalla ja vähentää nämä luvut ruudun keskipisteen koordinaateista. Seuraavaksi käydään kahdella sisäkkäisellä for-silmukalla koko taulukko läpi. Palikan x- ja y-koordinaatit saadaan laskettua hyödyntämällä silmukoiden juoksevia lukuja row ja col. Kertomalla jo aikaimmin laskettuja *blockWidth* ja *blockHeight* lukuja juoksevilla luvuilla col ja row saadaan palikka aina siirtymään jokaisella laskutoimituksella yhden palikan leveyden ja korkeuden verran ruudulla oikealle. Lisäämällä lukuihin vielä laudan sijainnin saadaan palikoiden koordinaatit vastaamaan haluttua ruudun keskipistettä.

8.3.4 Palikoiden piirtäminen

Seuraavaksi täytyi saada palikat piirtymään ruudulle, jotta voitiin todeta yllä olevan funktion toimivan. Piirtäminen libgdx:ssä tapahtuu käyttämällä libgdx:n omaa SpriteBatch-luokkaa. SpriteBatch hoitaa kaiken openGL:ään liittyvän toiminnallisuuden, joten siihen kehittäjän ei tarvitse puuttua ollenkaan, jos ei sitä halua. Jotta palikat saatiin piir-

rettyä ruudulle, täyty jokainen palikka taulukossa käydä läpi ja kutsua palikan omistamaa Sprite-oliota piirtoon. Seuraavana on eritetty palikoiden piirto.

Pääluokan render funktio

```
@Override
public void render () {
    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    batch.setProjectionMatrix(_camera.combined);
    batch.begin();
    for(int row = 0; row < BOARD_ROWS; row++){
        for(int col = 0; col < BOARD_COLUMNS; col++){
            blocks[row][col].drawBlock(batch);
        }
    }
    batch.end();
}
```

Block luokan funktio

```
public void drawBlock(SpriteBatch batch){
    _sprite.draw(batch);
}
```

Esimerkkikoodi 5 Palikoiden piirtäminen

8.3.5 Kosketuksen tarkistus

Seuraavaksi piti saada peli reagoimaan käyttäjän antamiin ruudun kosketuksiin. Libgdx:n tarjoamalla input moduulissa pystytään kysymään, onko käyttäjä esimerkiksi koskenut ruutua tai onko käyttäjä painanut jotain näppäintä näppäimistöltään. Jotta saatiin tietää, mitä palikkaa käyttäjä oli ruudulla painanut, täytyi tehdä yksinkertainen kosketuksen tarkistus funktio. Koska jokaisella palikalla omistaa rectangle-olion, joka pitää sisällään palikan koon ja sijainnin, pystytään siltä kysymään, sijaitseeko jokin tietty piste sen sisällä. Alla on esitettynä kosketuksen tarkistaminen.

Esimerkkikoodi 6 Palikoiden kosketuksen tarkistus

```
private void checkTouch(){
    if(Gdx.input.justTouched()){
        _touchPos.set(Gdx.input.getX(), Gdx.input.getY(), 0);
        _camera.unproject(_touchPos);

        for(int row = 0; row < BOARD_ROWS; row++){
            for(int col = 0; col < BOARD_COLUMNS; col++){

                if(blocks[row][col].getBounds().contains(_touchPos.x, _touchPos.y)){
                    blocks[row][col] = null;
                }
            }
        }
    }
}
```

Aluksi tarkastetaan, onko pelaaja edes painanut laitteen ruutua. Tähän käytetään input-moduulin tarjoamaa justTouched-funktiota, joka palauttaa joko true tai false. Jos näyttöä on painettu, laitetaan input-moduulilta saadut x- ja y-koordinaatit talteen Vector3 tyyppiseen olioon _touchPos. Vector3 sisältää kolme parametriä, jotka ovat x, y ja z. Tällaisella oliolla on siis helppo pitää sijaintia tallessa. Kun saatu sijainti on tallennettu, se annetaan pelin kameralle, joka muuttaa koordinaatit vastaamaan pelilaudan koordinaatteja. Koska input-moduuli antaa x- ja y-arvot, jotka vastaavat laitteen fyysisen resoluution koordinaatteja, joudutaan siksi tälläinen konversio tekemään. Seuraavaksi käydään kaikki palikat läpi taulukosta, ja kysytään palikan omistamalta Rectangle-oliolta, onko juuri saatu kosketus palikan sisällä. Jos näin on, kyseinen palikka poistetaan taulukosta.

Kuva 13 SumTowerin varhaista pelikuvaa

8.3.6 Palikoiden siirtäminen taulukossa

Palikan paikan vaihtuminen pelilaudalla oli seuraavana haasteena. Koska palikat oli päätetty sijoittaa kaksiuloitteiseen taulukkoon, helpottui tämän ongelman ratkaiseminen huomattavasti. Kun palikka poistetaan pelilaudalta koskettamalla ruutua, kyseinen taulukon solu, missä palikka sijaitsi, asetetaan tyhjäksi, niin kuin esimerkkikoodi 6:ssa havainnollistetaan. Jotta ylempi palikka voisi siirtyä alemmas, täytyy käydä läpi ensin kaikki solut taulukossa ja kysyä, onko tämä solu tyhjä. Kun tyhjä solu tulee vastaan, niin voidaan tarkistaa, onko ylemmässä solussa palikkaa. Jos ylemmästä solusta löytyy palikka, niin tämä palikka kopioidaan alempaan tyhjään soluun ja asetetaan vanha taulukon solu tyhjäksi. Seuraavana on esitetty palikan siirtäminen taulukossa.

```
private void moveBlocks(){
    for(int row = 0; row < BOARD_ROWS; row++){
        for(int col = 0; col < BOARD_COLUMNS; col++){
            if(blocks[row][col] == null){
                blocks[row][col] = getBlockAbove(row, col);
            }
        }
    }
}

private Block getBlockAbove(int startRow, int column){
    Block blockToMove = null;
    for(int row = startRow; row < BOARD_ROWS; row++){
        if(blocks[row][column] != null){
            blockToMove = new Block(blocks[row][column]);
            blocks[row][column] = null;
            break;
        }
    }
    return blockToMove;
}
```

Esimerkkikoodi 7 Palikoiden liikuttaminen taulukossa

Ensimmäinen funktio moveBlocks käy läpi kaikki solut taulukosta. Kun vastaan tulee tyhjä solu, niin siirrytään käyttämään apufunktiota getBlockAbove, jonka tehtävä on etsiä ensimmäinen mahdollinen siirrettävä palikka siltä sarakkeelta, mistä alun perin löytyi tyhjä solu. Tälle funktiolle annetaan parametreinä rivi ja sarake, mistä sen kuuluu lähteä lukemaan soluja ylöspäin. Kun funktio löytää solun, missä sijaitsee palikka, niin se tekee kopion kyseisestä palikasta käyttäen Block-luokan kopiokonstruktoria, asettaa vanhan solun tyhjäksi ja hyppää ulos for-loopista käyttämällä break-komentoa. Sen jälkeen se palauttaa löydetyn palikan, joka asetetaan tyhjään soluun.

8.3.7 Palikoiden yhdistäminen

Kun palikoiden siirto on tapahtunut, seuraavaksi tarkastetaan, sijaitseeko juuri siirtyneen palikan vieressä tai alapuolella samanvärisiä palikoita. Palikka-luokan getColor-funktiota hyödyntäen tämä vertailu voidaan suorittaa. Jos viereinen tai alapuolella oleva palikka on samanvärinen, niin siirtyneen palikan arvoon lisätään jokaisen samanvärisen palikan. Esimerkkikoodi 8 havainnollistaa palikoiden yhdistämisen.

```
private void checkMatchingColors(int row, int column){
    //block on left
    try{
        if(blocks[row][column].getColor() == blocks[row][column - 1].getColor()){
            blocks[row][column].addToValue(blocks[row][column - 1].getValue());
            blocks[row][column - 1] = null;
        }
    }catch (IndexOutOfBoundsException e){
        System.out.println(e + "there was no block to check on left side");
    }
    //block on right
    try{
        if(blocks[row][column].getColor() == blocks[row][column + 1].getColor()){
            blocks[row][column].addToValue(blocks[row][column + 1].getValue());
            blocks[row][column + 1] = null;
        }
    }catch (IndexOutOfBoundsException e){
        System.out.println(e + "there was no block to check on right side");
    }
    //block below
    try{
        if(blocks[row][column].getColor() == blocks[row - 1][column].getColor()){
            blocks[row][column].addToValue(blocks[row - 1][column].getValue());
            blocks[row - 1][column] = null;
        }
    }catch (IndexOutOfBoundsException e){
        System.out.println(e + "there was no block to check below");
    }
}
```

Esimerkkikoodi 8 Samanvärisien palikoiden yhdistäminen

Jokainen mahdollinen suunta käydään läpi käyttämällä try-catch metodia. Try-catch testaa halutun asian mahdollisten virheiden varalta, nappaa virheen, jos sellainen sattuu ja jatkaa ohjelman suoritusta. Jos mahdollisia virheitä ei testattaisi näin, niin ohjelma kaatuisi. Syy miksi tätä metodia joudutaan käyttämään palikoiden yhdistämisessä on se, että joudumme kysymään palikoiden taulukolta, onko esimerkiksi taulukon solussa [3][-1] olevan palikan väri sama kuin juuri taulukon soluun [3][0] siirtyneen palikan väri. Koska tiedämme että tästä tulee tapahtumaan virhe, sillä taulukossa ei ole solua

[3][-1], niin varmistamme ohjelman toimivuuden ja vältämme älyttömän määrän if-lauseita käyttämällä try-catch-metodia. Jokaisessa kohdassa kysytään ensin, vastaavatko palikoiden värit toisiaan. Jos värit ovat samat, niin palikkaan lisätään vieressä tai alapuolella olevan palikan arvo käyttämällä palikka-luokan funktiota addToValue, joka ottaa vastaan lisättävän palikan arvon, joka saadaan funktiolla palikka-luokan funktiolla getValue. Tämän jälkeen palikka, joka juuri lisättiin, poistetaan taulukosta asettamalle sen solu tyhjäksi.

8.3.8 Uusien palikoiden lisääminen pelilaudalle

Kun siirrot ovat tapahtuneet, täytyy vielä pelilaudan ylärivi täyttää uusilla palikoilla. Laudan täyttäminen tapahtuu hyvin samalla tavalla, kuin jo aikaisemmin esitetty palikoiden alustus taulukkoon. Jos joku taulukon ylärivin soluista on tyhjä, niin siihen asetetaan uusi palikka, jonka arvo on 1 ja väri on satunnainen. Esimerkkikoodissa 9 on esitetty uusien palikoiden lisääminen pelilaudalle.

```
private void spawnNewBlocks(){
    for(int col = 0; col < BOARD_COLUMNS; col++){
        if(blocks[BOARD_ROWS - 1][col] == null){
            float blockPositionX = (blockWidth * col) + boardPositionX;
            float blockPositionY = (blockHeight * BOARD_ROWS - 1) + boardPositionY;
            blocks[BOARD_ROWS - 1][col] = new Block(1, randomizeColor(),
                    new Rectangle(blockWidth,
                            blockHeight,
                            blockPositionX,
                            blockPositionY)
            );
        }
    }
}
```

Esimerkkikoodi 9 Uusien palikoiden lisääminen pelilaudalle

Koodissa käydään yksinkertaisesti vain taulukon ylärivi läpi for-loopilla ja tarkastetaan, onko jokin soluista tyhjä. Kun tyhjä solu löytyy, siihen asetetaan uusi palikka.

9 Pohdinta

Yksi tärkeimmistä tavoitteista projektia tehdessä oli kehittää omia taitoja niin ohjelmoinnissa yleisesti kuin mobiilipelien kehityksessä. Vaikka projektissa työni oli ohjel-

moida peliä, niin väistämättä tuli myös tilanteita, joissa pääsi miettimään pelin designia ja pelimekaniikkoja. Myös libgdx, joka oli jo entuudestaan tuttu viitekehys, avasi itsestään uusia puolia, joita pystyy käyttämään hyödyksi tulevissa projekteissa.

Ohjelmointitaitoni kehittyivät omasta mielestäni huomattavasti projektin aikana. Kun projekti aloitettiin, en ollut kovin varma omasta osaamisestani, sillä tämä projekti oli suurin, mitä koskaan olin ennen tehnyt. Ongelmanratkaisukykyni kehittyi huomattavasti ja omasta koodista tuli tehokkaampaa. Tätä raporttiakin kirjoittaessa ja koodia läpikäydessä huomasin siellä asioita, joita voisi tehdä paremmin. Yksi näistä huomioista on esimerkkikoodissa 7 esitetty tapa tarkastaa viereisiä palikoita. Aikaisemmin se oli toteutettu tekemällä monta peräkkäistä if-kyselyä, mutta kun koodia silmäili uudelleen, niin huomasin, että on tehokkaampikin tapa toteuttaa se. Myös taitoni lukea koodia on kehittynyt huomattavasti projektin aikana. Tässä projektissa on monia tuhansia rivejä koodia- ja koska se kaikki ei ole itse kirjoitettua, niin täytyy osta lukea, mitä jonkun muun tekemässä ongelman ratkaisussa tapahtuu.

Yhdeksi tärkeäksi kehityskohteeksi asetin itselleni projektin kokonaisuutena näkemisen ja sen pilkkomisen pienempiin osiin. Tässä on omasta mielestäni vielä kehityksen varaa, sillä välillä projektia tehdessä huomasin, että olin alkanut miettimään jotain ongelman ratkaisua liian laajasti. Loppujen lopuksi huomasin, että jos olisin ratkaissut ensin pienen osan ongelmaa, olisi kokonaisuuden ratkaisu helpottunut huomattavasti. Kehitystä tällä saralla tuli kiitettävästi.

Tiedonhankintataidot ovat kehittyneet huomattavasti. Kaikkia ongelmia kun ei itse voi ratkaista, niin välillä on turvauduttava tiedon etsintään ja katsoa, onko joku toinen painiinut samankaltaisen ongelman kanssa. Projektia tehdessä tuli vastaan monta uutta hyvää internetsivustoa, joista löytyi apua ongelmatilanteissa. Internet on siis hyvin tärkeä työväline ohjelmoinnissa. On hyvä oppia silmämääräisesti suodattamaan turha tieto, koska muuten voi mennä hyvin paljon turhaa aikaa hukkaan jossakin ongelmassa.

Mobiilipelien ohjelmoinnissa on omalta osaltani tapahtunut suurin kehitys. Se miten itse tämän huomaan, on, kun pelaan muiden tekemiä pelejä, joita esimerkiksi olen ladannut omaan älypuhelimeeni. Pelatessani mietin, miten joku pelistä löytyvä ominaisuus on tehty ja saatan jopa tehdä jonkun pienen testiprojektin, jossa koitan tuottaa tälläisen

samanlaisen featuren. Kiinnostus alaa kohtaan on kasvanut projektin myötä huomattavasti.

Lähteet

1 Mashable. 2016. Mobile Games. Verkkodokumentti. <http://mashable.com/category/mobile-games/> Luettu 5.2.2016.

2 Phone Arena. 2011. History of mobile gaming. Verkkodokumentti. <http://www.phonearena.com/news/History-of-mobile-gaming_id17949> Luettu 15.1.2016.

3 Java World. 2003. Develop state-of-the-art mobile games. Verkkodokumentti. <http://www.javaworld.com/article/2073796/mobile-java/mobile-java-develop-state-of-the-art-mobile-games.html> 7.11.2003. Luettu 3.2.2015.

4 Tom Warren. 2014. iPhone: A visual history. Verkkodokumentti. <http://www.theverge.com/2014/9/9/6125849/iphone-history-pictures>. Luettu 8.2.2016.

5 Cnet. 2015. Modern Combat 5: Blackout review. Verkkodokumentti. <http://www.cnet.com/products/modern-combat-5-blackout/>. Luettu 24.2.2016.

6 Superdata. 2015. Mobile games market. Verkkodokumentti. <https://www.superdataresearch.com/market-data/mobile-games-market/>. Luettu 26.2.2016.

7 Jesse Freeman. 2015. How to make a game part 1: Picking a framework. <http://jessefreeman.com/game-dev/getting-started-making-games-part-1-picking-framework/>. Luettu 26.2.2016.

8 Cocos2d-x. 2015. COCOS2D-X. Verkkodokumentti. < http://www.cocos2d-x.org/wiki/Cocos2d-x>. Luettu 1.3.2016.

9 Java. 2015. Learn About Java Technology. Verkkodokumentti. <http://java.com/en/about/>. Luettu 15.2.2016.

10 Oracle. 2015. About the Java Technology. Verkkodokumentti. <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT>. Luettu 15.2.2016.

11 Webopedia. 2015. OOP - Object Oriented Programming Definition. <http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html>. Verkkodokumentti. Luettu 18.2.2016.

12 Craig, Iain D. 2007. Object-oriented programming languages : interpretation. London : Springer.

13 Android developers. 2015. Android Studio Overview. Verkkodokumentti. <http://developer.android.com/tools/studio/index.html> Luettu 2.1.2016.

14 Github. 2015. Libgdx. Introduction. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Introduction> 29.7.2015. Luettu 2.1.2016.

15 Techopedia. 2015. Native code. Verkkodokumentti. <https://www.techopedia.com/definition/3846/native-code> Luettu 2.1.2016.

16 Github. 2015. Libgdx. The life cycle. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-life-cycle> Luettu 14.1.2016.

17 Github. 2015. Libgdx. The application framework. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-application-framework> Luettu 4.1.2016.

18 Chacon, Scott & Straub, Ben. 2014. Pro Git. Apress.

Android- ja työpöytäsovelluksen sekä pääohjelman käynnistysluokat

```
public class AndroidLauncher extends AndroidApplication {

    /**
     * Android-sovelluksen käynnistysluokka. Tänne tulevat kaikki sovelluksen
     * käynnistykseen liittyvät alustukset, jotka halutaan suorittaa Android alustoilla.
     *
     * @param savedInstanceState
     */

    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();
        initialize(new ExampleProject(), config);
    }
}
public class DesktopLauncher {

    /**
     *Työpöytäsovelluksen käynnistysluokka. Tänne tulevat kaikki sovelluksen
     *käynnistykseen liittyvät alustukset, jotka halutaan suorittaa työpöytäsovelluksilla.
     *
     * @param arg
     */

    public static void main (String[] arg) {
        LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
        new LwjglApplication(new ExampleProject(), config);
    }
}
public class ExampleProject extends ApplicationAdapter {
    /**
     * projektin pääluokka.
     */

    SpriteBatch batch;
    Texture img;

    @Override
    public void create () {
        batch = new SpriteBatch();
        img = new Texture("badlogic.jpg");
    }

    @Override
    public void render () {
        Gdx.gl.glClearColor(1, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        batch.begin();
        batch.draw(img, 0, 0);
        batch.end();
    }
}
```